Reinforcement Q-Learning using OpenAI Gym

University of Oulu
Faculty of Information Technology and Electrical Engineering
Degree Programme in Information Processing Science
Bachelor's Thesis
Juuso Laivamaa
12/03/2019

Abstract

Q-Learning is an off-policy algorithm for reinforcement learning, that can be used to find optimal policies in Markovian domains. This thesis is about how Q-Learning can be applied to a test environment in the OpenAI Gym toolkit. The utility of testing the algorithm on a problem case is to find out how well it performs as well proving the practical utility of the algorithm. This thesis starts off with a general overview of reinforcement learning as well as the Markov decision process, both of which are crucial in understanding the theoretical groundwork that Q-Learning is based on. After that we move on to discussing the Q-Learning technique itself and dissect the algorithm in detail. We also go over OpenAI Gym toolkit and how it can be used to test the algorithm's functionality. Finally, we introduce the problem case and apply the algorithm to solve it and analyse the results.

The reasoning for this thesis is the rise of reinforcement learning and its increasing relevance in the future as technological progress allows for more and more complex and sophisticated applications of machine learning and artificial intelligence.

Keywords
reinforcement learning, q-learning, openai gym, markov decision process, artificial intelligence

Ari Vesanen

Contents

Abstract 2
Contents 3
1. Introduction 4
    1.1 Research question 4
    1.2 Purpose 4
2. Reinforcement Learning 6
    2.1 Examples 7
    2.2 Policy 8
    2.3 Reward signal 8
    2.4 Value function 9
    2.5 Model 9
3. Markov Decision Process 10
    3.1 Markov property 10
    3.2 Markov chain 11
    3.3 Discount factor 12
4. Q-Learning 14
    4.1 Algorithm 14
5. OpenAI Gym 16
    5.1 Installing the toolkit 16
    5.2 Main elements 17
    5.3 Design 17
6. Problem Case 19
    6.1 CartPole Problem 19
    6.2 Adapting Q-Learning to the CartPole Problem 19
    6.3 Results 21
7. Conclusion 22
References 23

1. Introduction

Q-Learning is a reinforcement learning technique used in machine learning. It's described by Watkins and Dayan (1992, p. 1) as a form of model-free, or alternatively known as temporal difference learning. It can also be viewed as an asynchronous dynamic programming method. Barber (2012, Preface) describes that in general terms, machine learning as a field of study aims to learn useful information about the environment where a given organism operates. How the information is gathered then gives rise to the development of algorithms, which could be thought of as recipes on how to deal with uncertainty and complex data.

Russell and Norvig (2010, p. 10) describe reinforcement learning as an area of machine learning focused on studying and automating goal-driven learning and decision making. Reinforcement learning is commonly applied to problems concerning trial and error based learning and optimal control.

1.1 Research question

The primary research question for this thesis is how the Q-Learning algorithm can be applied to solve the CartPole problem, which is a classic control problem popularized by Barto, Sutton & Anderson (1983) and included in the OpenAI Gym library of test problems. Secondary aims for this study include explaining the foundational information that a proper understanding of the Q-Learning technique requires, such as knowledge about the basics of reinforcement learning, Markov decision process as well as how the algorithm works in theory and how it can be applied to a practical use case in a machine learning testing environment.

1.2 Purpose

The purpose of this thesis is to study how the Q-Learning algorithm works in theory as well as how it can be used to solve a problem case in a machine learning toolkits testing environment. However, before we can adequately understand how the Q-Learning algorithm functions we will first have to do the groundwork of going over the basics of the underlying systems and techniques that the algorithm is founded on.

Q-Learning is a reinforcement learning technique, so it logically follows that we first must go over what reinforcement learning is, how it works and how it relates to the subject of Q-Learning. Reinforcement learning is covered in the next chapter, where the aim is to get a general idea of subject which will help inform and ground the discussion of the later chapters.

Another subject that is crucial in understanding how Q-Learning works is the Markov decision process, which explains the logic of how the Q-Learning algorithm operates and chooses its policy outcomes. Markov decision process is covered in detail in the third chapter

After having laid the groundwork by covering the foundations that Q-Learning is based on, we will move on to dissecting how the Q-Learning algorithm actually works. We will start with a brief overview of the algorithm, where we will get a general idea of the process before delving into a more detailed explanation. In the detailed explanation we will go over the technique line-by-line by following and unpacking a pseudocode rendition of the Q-Learning algorithm. Q-Learning section

is covered in the fourth chapter and builds on the knowledge of reinforcement learning and Markov decision process chapters.

The fifth chapter provides a short overview on the OpenAI Gym toolkit, which is a machine learning environment that we will utilize when we introduce a problem case to the Q-Learning algorithm and find out how the algorithm can be applied to solve the problem in practice.

After we have covered the theoretical basis of this thesis which ends after we have an understanding of how Q-Learning and OpenAI Gym work, we will finally move on to the practical portion of this study where we introduce a problem case to the algorithm and see how we can apply it to the problem. OpenAI Gym has a built-in library that has a collection of numerous test problems or otherwise known as environments that can be used to develop and compare different machine learning algorithms and figure out how they match up in practical use to solve a problem. For this thesis we have chosen one the classic control theory problems, the CartPole problem. We will we have run the algorithm on the problem case and seen how it functions and go over and analyze the results and how well the algorithm performed. All of this will be covered in the sixth chapter.

The last chapter is the conclusion, where will summarize the information we have learned over the course of the study as well as discuss the limitations of the study and give recommendations for future study.

In the course of this thesis we will explain the basics of reinforcement learning, what it is and how it relates to the Q-Learning algorithm as well as the Markov decision process. We will also go over how the Q-Learning algorithm functions both theory and how it can be used to solve practical test problems by using the OpenAI Gym toolkit, which is machine learning environment for testing reinforcement learning algorithms.

2. Reinforcement Learning

According to the standards of International Organisation for Standardization's guideline for artificial intelligence and machine learning (ISO/IEC JTC 1, 1997), reinforcement learning is defined as "learning improved by credit and blame assignment". Reinforcement learning is an area of machine learning and is described by Francois-Lavet, Henderson, Islam, Bellemare, & Pineau (2018, p. 1) as a process of figuring out how learning agents ought to choose a sequence of actions in a given environment in order to maximize cumulative numerical rewards.

What makes reinforcement learning unique is that the learner is not told explicitly what actions they must take, but must instead discover independently which actions produce the highest rewards by trying them over and over again. Sutton and Barto (2017, p. 1) posit that in the most interesting and complex cases the actions committed by the learning agent may not only affect the immediate reward produced by said action, but can also have repercussions for future situations and future rewards. Sutton and Barto (2017, p. 1) go on to mark out from this two primary characteristics that are the most important features of reinforcement learning, search by trial-and-error and delayed rewards. These two qualities differentiate reinforcement learning from other machine learning paradigms. For example, Francois-Livet et al., (2018, p. 15) compare reinforcement learning's trial-and-error experience to dynamic programming that assumes perfect information of the environment by default.

Sutton and Barto (2017, p. 1-2) go on to formalize the problem of reinforcement learning as the process of finding an optimal way to control Markov decision processes with incomplete information about the environment. We won't go into the details of the Markov decision process here, as it will be discussed in more detail in the next chapter. What's trying to be accomplished in reinforcement learning is the capturing of vital information about a problem that the learning agent is facing while interacting with its environment over time. For this to be possible, the learning agent must be able to in some way sense the environment and what state it's currently in as well as being able to take actions that can modify that state in a non-trivial way. Finally, the agent must also have a goal or goals relating to the environment that can be completed by taking actions that are possible for the agent.

There are also two other paradigms of machine learning: supervised and unsupervised learning, which are both different from reinforcement learning. Supervised learning is described by Russell and Norvig (2010, p. 695) as one the three main types of learning, where the learning agent observes given input-output pairs known as training sets and learns a function that maps in between them. The training sets are given by a knowledgeable supervisor, commonly a human tester, who is external to the environment and to the agent. Each training set includes a description of a situation as well as a label that specifies the correct action. The purpose of this kind of learning is to teach a system to be able to learn the correct actions from the training sets and generalize that information so that it's able to act correctly in new situations that aren't described in the training sets. The other type, unsupervised learning, excludes both the role of the supervisor and labeled data, dealing instead with collections of unsorted data from which it tries to find hidden structures and connections. Francois-Livet et al., (2018, p. 9) relates it to identifying patterns and using them for tasks such as generative models and data compression.

Sutton and Barto (2017, p. 2) note that from a quick glance, it might occur to some that reinforcement learning itself fits into the unsupervised learning paradigm, however while uncovering hidden structures is certainly a key part of reinforcement learning, it doesn't address the main problem of reward signals that reinforcement learning is concerned with. This is the reasoning why they consider reinforcement learning the third machine learning paradigm, separate from both supervised and unsupervised learning.

A central problem that occurs in reinforcement learning that doesn't arise in other kinds of learning is the trade-off that the learning agent makes when exploring and exploiting the environment. By exploitation we mean the act of maximizing the expected return value of an action and by exploration we mean the obtainment of new information about the environment. Francois-Livet et al., (2018, p. 9) argues that the problem occurs because the imperative for the agent to maximize reward given its current knowledge of the environment forces the agent to prefer actions that it has taken in the past that have proven to be the most effective in returning the greatest value, which in turn means eschewing the option of exploring more of the unknown environment. This means that the agent cannot explore new areas in order to make better action selections that return higher rewards in the future. This is what is commonly known as the exploration/exploitation dilemma, which simply stated means that neither exploration or exploitation can be pursued exclusively without the failing the task of maximizing reward. Despite mathematicians having intensively studied the dilemma for many decades, it still remains unresolved as of this day (Sutton & Barto 2017, p. 2).

Sutton and Barto (2017, p. 2) continue explaining that another key feature that makes reinforcement learning unique among its peers is its explicit focus on considering the whole problem of a goal-driven learning agent in an imperfect knowledge environment. This is in contrast to many other approaches, which deal exclusively with subproblems, without concerning themselves with the problem of how they interact in the larger picture. That is not to say that these approaches are inferior or wrong, they indeed have produced many very important and useful findings, however their limited focus is a disadvantage that isn't shared by reinforcement learning.

Reinforcement learning starts with a completely formed and interactive goal-seeking agent, which has explicit goals, can sense the state of its environment and can choose actions that can influence that environment. (Sutton & Barto 2017, p. 3). It's also assumed that agent has to start in the beginning with varying degrees of uncertainty about the state of its surrounding environment. Nature of the agent can also vary wildly depending on the environment, for example it might mean an autonomous entity like a robot or even a small component of a larger system, such as a temperature regulator in a complex machine.

2.1 Examples

Apart from knowing the minutiae of the principles that reinforcement learning is built on, a short look at concrete examples of reinforcement learning in practical applications is a good way to get a clear picture of what is and isn't reinforcement learning.

Silver (2015, Lecture 1, p. 9) in his UCL course on reinforcement learning gives short examples of reinforcement learning applications. One is managing a power station, in which the learning agent is a component of the power station's larger system which controls its operations. Here the agent is an adaptive controller of the system, where its goal is to optimize the operation of the power station and by doing so maximizing the return value or the output of the process. In order to do so, the agent must *explore* its environment, by observing the function of different components and

processes of the system and *exploiting* it by adjusting the environment to the most optimal as possible settings, which yield the largest output of cumulative rewards.

Another interesting example provided by Silver (2015, Lecture 1, p. 9) is defeating the world champion at Backgammon, which is great illustration of how the agent must consider both the immediate rewards of their next move on the board as well as how it might affect the cumulative rewards acquired in the next possible sequence of moves. The choice that the agent makes is informed both anticipating possible replies and countermoves of the other player as well as by immediate and intuitive judgements about the value of particular moves and position on the board.

These two examples share basic features of reinforcement learning that are easy to overlook. They both involve active decision-making agents interacting between their environment, where an agent has a goal that it is trying to fulfill in an environment that it doesn't have complete knowledge of. The agents are also permitted to make choices that affect the future state of the environment and which can influence delayed rewards in the next sequence of moves. The long term effect of actions taken can't be reliably predicted, which means that the agent must monitor the change in its environment on a frequent basis and make appropriate changes to its actions in order to keep up. Both of these examples also feature explicit goals, for the controller it is the optimization of the power stations functions and for the Backgammon-playing agent it's defeating the world champion. They also require experience over time to improve performance. In the process of controlling the power station or playing Backgammon the agent refines their understanding of their environment which they use to evaluate their current situation and predict future consequences of their actions. This means incremental improvement over time as an agent finds new ways to optimize and streamline its actions.

2.2 Policy

Sutton and Barto (2017, p. 5) highlight that beyond the agent and the environment we have already discussed before, one can identify four other sub-elements of reinforcement learning systems: *policy*, *reward signal*, *value function* and optionally a *model* of the environment. First of these we will discuss is policy, by which we simply mean the agent's behavior (Silver, 2015, Lecture 1, p. 26). Policy, roughly speaking, is the mapping from the perceived state of the environment to an action that should be taken when in that state. Sutton and Barto (2017, p. 5) equate it to what in psychology is known as a set of stimulus and response associations. Depending on the context, the policy can be something relatively simple, such as a short function, whereas in more complex situations it could involve more performance intensive computation such as a search process.

Policy can be thought of as the core of a learning agent, since it alone is sufficient to determine a behavior (Sutton & Barto, 2017, p. 5). Policies can be either deterministic, where the state of the environment is known, or stochastic, where the environment is random or filled with uncertainty. Policies can also be categorized by a second criterion, stationary or non-stationary. A non-stationary policy depends on time-steps and is useful for finite-horizon contexts where the cumulative rewards that the agent seeks to optimize are limited to a finite number of future actions (Francois-Livet et al., 2018, p. 22).

2.3 Reward signal

Sutton and Barto (2017, p. 5) describe reward signal as defining the goal of any given reinforcement learning problem. On each time step the environment sends the agent a numerical value known as the reward, which the agent notes as it tries to maximize the amount of reward it attains in the long run.

The reward signal defines good and bad outcomes for the agent, and thereby signals which actions are preferable to others by the amount of their return value. Sutton and Barto (2017, p. 5) characterize this as being akin to a biological system in humans, where the feelings of pleasure and pain signal to the human brain that the value of putting your hand on a hot stove is lower than eating a delicious meal. Altering a policy is founded primarily on what the reward signals inform the agent about the environment, if an action selected by the policy returns a low value reward then the policy can be changed to select some other action in the same situation in the future. Sutton and Barto (2017, p. 5) point out that in general, reward signals can be thought of as stochastic functions about the state of the environment and the action taken by the agent.

2.4 Value function

Sutton and Barto (2017, p. 5) compare reward signals and value functions by saying that whereas reward signals can be thought of as an indication of what is good in the immediate situation, the value function specifies what is optimal in the long run. Generally speaking, the value of any given state is the total amount of value that the agent can expect in the future, starting from that state. In contrast to rewards, which describes the current desirability of environmental states irrespective of future events, the value function takes into account the long term picture of the desirability of states when considering the value of future states.

This means that the value function factors in circumstances where there is the possibility of low immediate reward by picking a state, but high reward that can be attained in the long run. Value functions also recognize situations where the reverse is true. Sutton and Barto (2017, p. 5) continue using a human analogy like in the previous subchapter by saying that rewards are similar to pleasure (high reward) and pain (low reward), whereas value functions can be equated to a proper and farsighted view on how short term pain can result in long term benefit, like in exercise and how short term pleasure can have negative long term effects, like obesity caused by overeating.

2.5 Model

The final sub-element of reinforcement learning is the model of the environment, which mimics the environment and that can be used to make educated guesses or inferences about how the environment will behave in different states. Sutton and Barto (2017, p. 5-6) give an example where given a state and an action, the model uses them to predict what resulting state and reward they will produce.

The main utility of models is planning, which means deciding a course of action and predicting the resulting scenario, changed state of the environment and the return value before it has been experienced. Sutton and Barto (2017, p. 5-6) inform that methods that are used in solving reinforcement learning problems which include planning and use of models are called model-based methods. These are in contrast to model-free methods of reinforcement learning, which revolve solely around trial-and-error learning, the opposite of planning.

3. Markov Decision Process

The problem of optimal sequential decision making has been studied since World War II, when the topic was pursued in the field of operations research in Britain, with the primary aim of optimizing radar installations in military use. After the war the research found civilian applications and was formalized into a class of sequential decision problems called Markov decision processes by Richard Bellman in 1957 (Russell & Norvig, 2009, p. 10).

Sigaud and Buffet (2010, p. 4-5) start out by defining Markov decision processes as controlled stochastic processes (as opposed to deterministic processes that form the traditional approaches to planning) that satisfy the Markov property and assign rewards to state transitions. Konstantopoulos (2009, p. 1) states that the time of the MDP can be either discrete (integers), continuous (real numbers) or even a totally ordered set. Markov decision process is also a dynamical system, meaning that in mathematical terms it's a phenomenon which evolves over time in a way where only the present affects the future state of the system.

Often in decision analysis we are focused on decision making in the face of one uncertain future event. However, in many cases involving decision making we need to take into account the uncertainty of many future events in succession in which case Markov decision processes are of great help. MDP's are used mainly to study and solve optimization problems in dynamic programming and reinforcement learning. This is because the main problem for MDP's is figuring out the optimal policy (a policy that yields the highest expected value) for a decision maker, which tells it what actions to take when in any given state.

Poole and Mackworth (2017, pp. 399-400) define a Markov decision process as a 5-tuple, which consists of:

- S, which is a finite set of possible world states,
- A, which is a finite set of possible actions,
- P: $S \times A \times A \rightarrow [0, 1]$, which specifies the dynamics of states and actions. This is commonly written as P(s'|a), where

  $\forall s \in S \quad \forall a \in A \quad \sum_{s' \in S} P(s'|s,a) = 1.$

  More specifically, P(s'|a) denotes the probability of transitioning to a state *s'* given that an action *a* is taken in a state *s*.

- R: $S \times A \times S \rightarrow \mathbf{R}$, where R(s, a, s') returns the expected immediate reward from doing action *a* and transitioning from state *s* to state *s'*.
- γ, which is a discount factor, where $\gamma \in [0, 1]$ presents the value of future rewards.

It is worth noting that while the theory of Markov decision processes doesn't explicitly state that states and actions are finite, most algorithms that use MDP's assume that they are.

3.1 Markov property

When it is said that a process satisfies the Markov property we mean that the effects of an action taken in a state are only contingent on that state and not on any other previous state. This is because

the current state captures all the relevant information about the past and is sufficient for making predictions about the future. Silver (2015, Lecture 2, p. 4) delivers a useful definition, which states that a state S_t has the Markov property only if:

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]. \quad (1)$$

Apart from sequences occupying some state in the environment, they are also there at a specific time so S_t is used to represent the state at time *t*. As a side note, it is sometimes also said that S_t is Markov instead of saying that it has the Markov property.

3.2 Markov chain

Often times an agent has to make predictions about an ongoing process where it doesn't know how many actions are required to accomplish a goal. This is what is called an infinite horizon problem, where process might go on forever or it can alternatively be an indefinite horizon problem, where there is an end point for the agent, but the agent doesn't know when it will occur. In order to model these situations, we need to fill the Markov chain with actions. Poole and Mackworth (2017, p. 399) state that a Markov decision process can be seen as a Markov chain filled in with rewards and actions or alternatively as an extended time decision network. The agent decides on an action to perform at each stage of the process and the reward and the transition state depend on the previous state and performed action.

Poole and Mackworth (2017, p. 266) give a definition of the Markov chain as a unique type of belief network that is used to represent sequences of values, like a set of states in a system or a set of words in a sentence. Each step in a Markov chain is a called a *stage*. Grinstead and Snell (2006, p. 405) describe a generic Markov chain as a set of states S $\{ s_1, s_2, \dots, s_r \}$. The process starts from one of the states in the Markov chain and moves sequentially from one state to another, which is called a *step* in this context. If a Markov chain is in a state a_i, then it moves to a next state a_j with a probability of p_{ij}. Note that the probability is not contingent on previous states that the agent was in before the current state as the Markov property is assumed. The probabilities denoted by p_{ij} are called the transition probabilities. Apart moving from one state to another, the process can also stay in its current state, which is denoted by the probability p_{ii}. The starting distribution of probabilities is defined usually as an *S*, which is also the starting state.

Poole and Mackworth (2017, p. 266) state that a Markov chain can be either a stationary model or a time-homogenous one depending on if all variables share the same domain, and the transition probabilities are identical for each stage of the Markov chain, i.e.:

$$\text{for all } i \geq 0,\ P(S_{i+1} \mid S_i) = P(S_1 \mid S_0) \quad (2)$$

For a stationary Markov chain, there are only two conditional probabilities that are provided, which are $P(S_0)$, which defines the initial conditions and $P(S_{i+1} \mid S_i)$, which qualifies the dynamics. Poole and Mackworth (2017, p. 266) note that stationary Markov chains provide a very simple model that is easy to use and that also assumes that the dynamics of the environment won't change over time and if they do it is usually because of some other element that could be modeled. Stationary model networks can also be extended indefinitely, which allows one to make observations or queries about any point in the past or the future.

Konstantopoulos (2009, p. 2) gives an illustrative example of a Markov chain by describing a scenario of a mouse in a cage. The mouse occupies a cage with two cells, A and B, which contain fresh and rotten cheese, respectively. An observing scientist's job is to record the position of the

mouse every minute. When the mouse is in the cell A at a given time *n* (minutes) then it follows that at time *n* + 1 it is still in cell A or has moved to cell B.

Observing the mouse's behavior statistically, the scientist comes to believe that the mouse moves from cell A to B with a probability of $\alpha = 0.05$. This is regardless of the cell the mouse was in the past. Conversely, the mouse moves from cell B to A with a probability of $\beta = 0.99$.

This information can be summarized with a transition diagram (Konstantopoulos, 2009, p. 3):

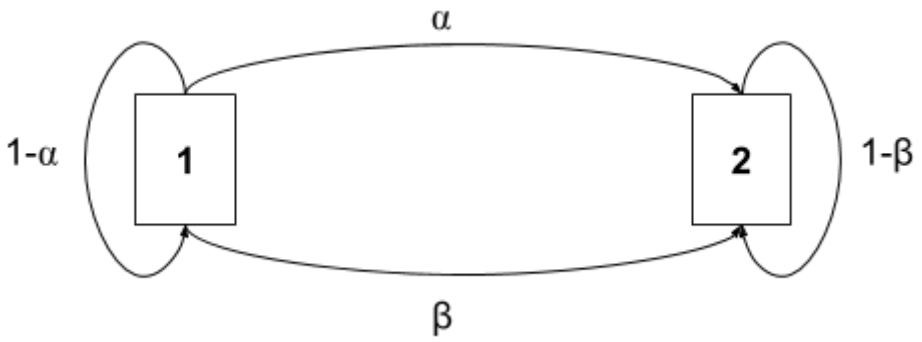

Figure 1. Transition Diagram (Konstantopoulos, 2009)

Alternatively, it can also be illustrated with a state transition matrix, where:

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} = \begin{pmatrix} 0.95 & 0.05 \\ 0.99 & 0.01 \end{pmatrix}$$

Figure 2. Transition Probability Diagram (Konstantopoulos, 2009)

Given this information, how long does it take on average for the mouse to move from cell A to cell B? Konstantopoulos (2009, p.3) answers that question by stating that since the mouses decision to stay or move to another cell is a coin flip, the first time the mouse moves from cell A to B will have a mean of $1/\alpha = 1/0.05 \approx 20$ minutes, which is the mean of the binomial distribution with the parameter α.

3.3 Discount factor

For a Markov decision process to perform optimally in the long run, it needs to consider how to weigh the differences between immediate rewards and future rewards. The way that MDP's go about doing this is by utilizing what is called a discount factor, which is represented by γ. Silver (2015, Lecture 2, p. 12) explains that the discount factor $\gamma \in [0, 1]$ is used to determine the value of future rewards, by picking a factor between 0 and 1. Gamma that is close to 0 leads to what is a called a "myopic" evaluation, which considers only short-term rewards and a γ that is close to 1 leads to a "far-sighted" evaluation, which strives for long-term rewards instead.

Matiisen (2015) points out that picking out the right discount factor is highly situationally dependant. For example, if we aren't sure about the value of future rewards and want a balanced consideration of both immediate and future rewards, we can set the discount factor to something like $\gamma = 0.9$. On the other hand, if all sequences are the same as in a deterministic environment and

actions always result in the same rewards, then we can forgo discounted rewards entirely (i.e. $\gamma = 1$).

4. Q-Learning

Q-Learning is an off-policy temporal difference learning algorithm for reinforcement learning, the goal of which is to learn a policy, which tells the agent what actions to take under what circumstances. Q-Learning is suited for cases where there isn't any explicit model of the system or the cost structure, which is why it's often called a model-free method of reinforcement learning. Jin, Allen-Zhu, Bubeck, & Jordan (2018, p. 1) point out that model-free methods are typically more flexible to use and easier to understand and thus are more prevalent in modern RL than model-based approaches. The advantages of model-free approaches like Q-Learning are multifaceted, they are online for one, require less space and are typically more expressive since specifying the policies and value functions are more flexible than specifying the model of the environment. Given sufficient training, a Q-Learning algorithm converges with a probability of 1 to closely match the action-value function for any arbitrary target policy. Q-Learning can learn the optimal policy even when actions are chosen according to a random or exploratory policy.

Watkins and Dayan (1992, p. 1) elaborate by saying that Q-Learning provides agents the capability of learning optimally in Markovian domains by experiencing the consequences of their actions and not requiring them to build maps of the domains. Bertsekas (2011, p. 326) adds that the aim of Q-Learning class of algorithms is to compute the optimal cost function for all states, not just the cost function of a single policy. It does this by updating the Q-factors associated with the optimal policy, instead of approximating the cost function of any particular policy. By doing this it neatly avoids the multiple policy evaluation steps of the policy iteration method.

The process of learning starts by the agent trying an action at a given state and evaluating the consequences of that action in terms of immediate rewards or penalties, which are received in the form of numerically assigned rewards and evaluated by estimating the value of the state which the agent has chosen. The agent continues trying out all actions in all states repeatedly and by doing so learning which states contain the highest return value overall, as judged by long term discounted rewards (Watkins & Dayan, 1992, p. 1). Q-Learning is a primitive form of learning, but can operate as a basis for higher level processes.

4.1 Algorithm

The procedural form of the algorithm is as follows (Vilches, 2019, Tutorial 1: Q-Learning):

```
Initialize Q(s,a) arbitrarily
Repeat (for each generation):
      Initialize state s
      While (s is not a terminal state):
            Choose a from s using policy derived from Q
            Take action a, observe r, s'
            Q(s,a) += α * (r + γ * maxQ(s') - Q(s,a))
            s = s'
```

in which:

- s is the previous world state.
- a is the agent's previous action.
- Q() is the Q-Learning algorithm.
- s' is the current world state.
- α is the learning rate, which is generally set to a number between 0 and 1. Setting it to 0 means that the Q-values are never updated and nothing new is learned. Setting alpha to a high value such as 0.9 means that the learning will occur very quickly.
- γ is the discount factor, which is also set between 0 and 1. This models the value difference between immediate and future rewards.
- max is the maximum reward that is attainable in the next world state.

Vilches (2019, Tutorial 1: Q-Learning) points out that the algorithm can also be understood in plain terms in the following way:

1. Initialize Q(s, a), which is the Q-values table. Q-Table is just a term for a simple lookup table where the maximum expected rewards for actions in each state are calculated.
2. Observe the current world state, s.
3. Decide on an action, a, for the world state s based on the agent's selection policy.
4. Take the action and observe the reward, r, as well as the new world state, s'.
5. Update the table's Q-value for the newly-observed reward and the maximum possible reward for the next world state.
6. Change the world state to the new state and repeat the process as long as a terminal state hasn't been reached.

The mathematical foundation of Q-Learning rests behind what are called Bellman equations, named after Richard E. Bellman. The structure of the Bellman equation is simply put that the current value equals the sum of immediate reward and future value when both are maximized. Here we will limit ourselves only to examining simple Q-Learning where the Q-function maps state-action pairs to maximized reward signals that are calculated by weighing immediate rewards with future returns. This is further qualified by other parameters like the learning rate and the discount factor. By quantifying these factors into a equation, we get something that is structurally not too dissimilar to the Bellman equation (Bennett, 2016):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3)$$

Going over the equation from left to right, we can see we are starting with the Q-function that we are updating, based on the state *s* and the action *a* at the time *t*. The arrow operator right after the function means that the Q-function will be updated to the left. In this case we add both the old and the new action values to the existing Q-value. After that is the learning rate, signified by α. The learning rate multiplies the product of new information - old information. r_{t+1} is the reward earned from transitioning from time *t* to the next turn *t+1*. γ is the discount rate, which determines how much the future rewards are worth. $\max Q(s_{t+1}, a)$ is the value of the action that is determined to return the maximum amount of total future reward that is then subtracted by the existing estimate of the Q function in its current state. Bennett (2016) provides also a simplified notation of the equation, which can be put as simply as: (The New Action Value = The Old Value) + The Learning Rate * (The New Information - The Old Information).

5. OpenAI Gym

Gym is a toolkit used for developing and testing reinforcement learning algorithms, created by OpenAI, a non-profit AI research company. OpenAI's mission is to discover and build a safe path to artificial general intelligence (AGI). They house a full-time staff of 60 researchers and engineers, focused on long-term research and being on the forefront of the advances of AI research (N/A, n.d., OpenAI Gym Documentation). OpenAI Gym makes no assumptions about the structure of the agent, and has compatibility with any numerical computation library, such as Theano or Google's Tensorflow. Gym is a library, which contains a collections of test problems, alternatively known as environments, which can be used test reinforcement learning algorithms. The environments share a common interface, which allows the user to write general algorithms.

A commonality in all of reinforcement learning is an agent situated in an environment. In each step, the agent takes an action and as a result receives an observation and a reward from the environment. Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, & Zaremba (2016, p. 1) point out that all reinforcement learning algorithms seek to maximize some measure of the agent's reward while the agent interacts with its environment. In literature, the environment is characterized in the form of a partially observable Markov decision process. What makes OpenAI Gym unique is how it focuses on the episodic setting of reinforcement learning, where the agent's action chains are broken down into a sequence of episodes. Each episode begins by randomly sampling the agent's initial state and continues until the environment reaches a terminal state. The purpose of structuring reinforcement learning into episodes like these is to maximize the expected total reward per episode, and to manage a high level of performance in as few episodes as possible.

5.1 Installing the toolkit

To install the OpenAI Gym you'll first need to have Python 3.5 or above installed on your computer. Python 3.5+ Documentation has a beginners guide page that has instructions on how to download Python and how to use it. Once you have Python, Gym can be installed simply by using the Python's package management system *pip*, like so (N/A, n.d., OpenAI Gym Documentation):

```
pip install gym
```

A good way to illustrate the basic structure of OpenAI Gym is first by writing the bare minimum of code to get something working. OpenAI Gym documentation presents a simple example of this by running an instance of CartPole-v0 -problem environment for a thousand time-steps and rendering the environment at each step:

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
```

As is visible from the above code, before we do anything using OpenAI Gym, we first need to import the gym library which contains all of the environments and dependencies that you need to use the toolkit. After that we create a variable for the environment, env, and set it to the environment we want to use, which we reset to the initial state. Once the selected environment is set

up, we create a for-loop for a thousand timesteps. In each loop we render the current state of the environment to the window and take a random action in the environment. The result is not anything useful as far as maximizing total rewards per episode, but it illustrates the base mechanics of OpenAI Gym in play.

When you run the code you should see a window pop up, which provides a visual render of the process:

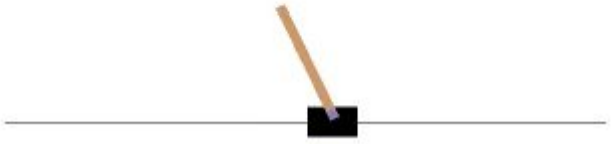

Figure 3. CartPole-v0 (OpenAI Gym Documentation, n.d.)

As a side note, the CartPole-v0 problem environment we have shown here will also be used for the problem case in the next chapter.

5.2 Main elements

The above code example shows how we can take random actions in each step, but if we want to know how to take better and more useful actions, we have to know how our actions are affecting the environment. OpenAI Gym Documentation (n.d.) presents us the environment's step -function, which returns four values, which are:

- Observation (object), which is an environment-specific object that represents the agent's observation of the environment. This can be something like the board state in Chess or the explored areas in a maze.
- Reward (float), which is the amount of reward returned by the previous action.
- Done (boolean), which represents the terminal state in an environment and tells the algorithm whether it's time to reset the environment again. Most tasks in Gym are separated into distinct episodes, and done being True means that the episode has reached a terminal state.
- Info (dict), which is a diagnostic tool used for debugging.

OpenAI Gym points out that the step -function is just an implementation of the classic "agent-environment" loop from RL. In each timestep, the agent takes an action upon which the environment returns an observation and a numerical reward.

5.3 Design

Brockman et al., (2016, p. 2) give a summary of the design decisions taken by the developers behind OpenAI Gym, which were based on their own experiences developing and comparing different reinforcement learning algorithms and using previous benchmarking collections. The first one of the decisions taken was prioritising environments over agents, by providing an abstraction only for the environment and not the agent. This choice was made in order to maximize user convenience and allowing them to independently implement different styles of agent interface. Second choice taken was to emphasize sample complexity and not just focusing on final

performance of algorithms. The performance of reinforcement learning algorithms can be measured two different ways, first is by the final performance (average reward per episode after learning is complete) and second by the the amount of time it takes for the agent to learn (sample complexity). The third choice made by the developers was encouraging peer review over competition, by allowing users to compare the performance of their algorithms on the OpenAI Gym website. This was done in order to encourage the users to share their code and ideas with others and to act as a meaningful benchmark for comparing different methods. The last choice noted by Brockman et al., (2016, p. 2) was strict versioning for environments. This means that if an environment changes, the results between previous and future versions of the environment would be incomparable. To avoid this problem, OpenAI team decided to guarantee that each update of the environment would be accompanied by an increase in the version number. For example, an update to CartPole-v0's functionality would mean that the new version would be called something like CartPole-v1.

6. Problem Case

Now that we have covered the theory behind the Q-Learning algorithm and the OpenAI Gym learning testing toolkit, we will move onto the central problem of how the Q-Learning algorithm can be implemented to solve the CartPole-v0 test problem.

6.1 CartPole Problem

The documentation page for CartPole-v0 (n.d.) describes the problem as consisting of a pole attached to a cart, which moves along a frictionless track in a two-dimensional environment. The system can be controlled by applying a force of +1 or -1 to the cart, which controls the pendulum shift of the pole to left or right. The pole starts in the upright position and the goal is to keep it from falling over. A numerical reward of +1 is returned for each time-step where the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical or the cart moves more than 2.4 units from the center. The environment is considered to be solved if an algorithm gets an average reward of 195.0 over 100 consecutive trials.

6.2 Adapting Q-Learning to the CartPole Problem

Vilches (2019, Tutorial 4: Q-learning in OpenAI gym) provides an example of how Q-Learning can be adapted to build a solution to the CartPole-v0 problem. We will now summarize the gist of how his code works:

```
if __name__== '__main__':
      env = gym.make('CartPole-v0')
      …
      qlearn = QLearn(actions=range(env.action_space.n),
                      alpha=0.5, gamma=0.90, epsilon=0.1)
```

In the above code we start off the code execution in the main -function by first setting the environment to the CartPole-v0 -test environment. Next we start the Q-Learning process by making a variable qlearn of the class QLearn, which we initialize with custom parameter values. The QLearn class takes parameters for actions, the discount constant alpha, the discount factor gamma and the exploration constant epsilon.

Next up we run the algorithm for 3000 episodes:

```
for i_episode in xrange(3000):
	observation = env.reset()
	…
	cart_position, pole_angle, cart_velocity, angle_rate_of_change = observation
	…
	for t in xrange(maximum_number_of_steps):
		# Pick an action based on the current state
		action = qlearn.chooseAction(state)
		# Execute the action and get feedback
		observation, reward, done, info = env.step(action)
		…
		if(not done):
			qlearn.learn(state, action, reward, nextState)
			state = nextState
		else:
			reward = -200
			qlearn.learn(state, action, reward, nextState)
			last_time_steps = numpy.append(last_time_steps, [int(t + 1)])
			break
```

The code execution starts off each episode by resetting the environment to the initial state and saving it in the observation variable. The information from the observation of the current environment is then used to update the current information on the status of the cart and the pole and the state of the environment as a whole. Next the code enters another for-loop for the maximum number of steps which in this case is 200. In each loop we pick an action based on the current state, which is done by calling the chooseAction() -function of the QLearn -class with the current state as a parameter. The code for the function looks like this:

```
def chooseAction(self, state, return_q=false):
	q = [self.getQ(state, a) for a in self.actions]
	maxQ = max(q)

	if random.random() < self.epsilon:
	minQ = min(q); mag = max(abs(minQ), abs(maxQ))
		# add random values to all the actions, recalculate maxQ
		q = [q[i] + random.random() * mag - .5 * mag for i in range(len(self.actions))]
		maxQ = max(q)

	count = q.count(maxQ)
	# In case there are several state-action max values
	# we select a random one among them
	if count > 1:
		best = [i for i in range(len(self.actions)) if q[i] == maxQ]
		i = random.choice(best)
	else:
		i = q.index(maxQ)

	action = self.actions[i]
	if return_q: # if they want it, give it!
		return action, q
	return action
```

After the action has been executed the feedback is saved in the observation, reward, done and info variables. If the done boolean returns false, which means that the episode hasn't reached a terminal

state, the learning process is continued changing the current state to the next state and by calling the class function learn() with the requisite parameters. The learn() -function looks like this:

```
def learn(self, state1, action1, reward, state2):
      maxqnew = max([self.getQ(state2, a) for a in self.actions])
      self.learnQ(state1, action1, reward, reward + self.gamma*maxqnew)
```

In the function the new maximum reward is obtained and learnQ() -function is called to figure out the table's Q-values for the newly-observed reward and the maximum possible reward for the next state. If done returns true, the reward is set to -200 and the maximum reward for the last state is obtained by calling the learn() -function and the last time-steps are recorded before breaking the loop and either starting a new episode or exiting the program. The code covered here is only a fraction of the full source code of the implementation that can be found on Vilches' Github page.

6.3 Results

The results saw substantial improvements over early generations, breaking the threshold for learning performance required in order to be seen as having successfully solved the problem by episode the 600th generation, at which point the rate of improvement between generations dips and recovers but doesn't see substantial improvement anymore before the algorithm reaches the end.

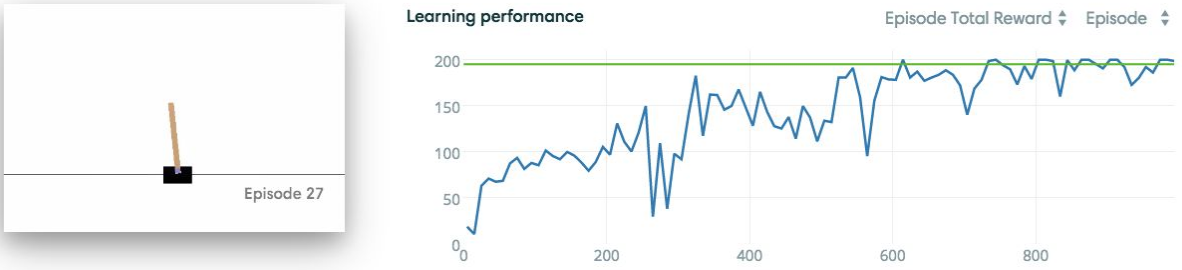

Figure 4. CartPole-v0 results. (Vilches, 2019)

The resulting performance can be seen as a success, as the algorithm passed the requirement of getting an average reward of 195.0 over 100 consecutive episodes after the algorithm was run through 1000 generations.

7. Conclusion

This thesis focused on the research question of how the Q-Learning algorithm can be used to solve a test problem in the OpenAI Gym library. To fulfill this task we first had to investigate on what premises was the algorithm's functionality founded on. Q-Learning is a reinforcement learning technique which meant that in order to understand it we first had to understand reinforcement learning on a general basis. Additionally, a basic knowledge of the Markov decision process was also required, which meant that had to be covered as well. After the foundational knowledge was unpacked, the functionality of the Q-Learning algorithm itself was next as well as how the OpenAI Gym toolkit could be used to test the performance of the algorithm. Lastly we finally moved on the problem case itself, which was chosen to be the CartPole-v0 -test problem. The implementation that was chosen was from Vilches' (2019) reinforcement learning tutorial concerning Q-Learning and was covered in a summarized form. The learning performance fulfilled the passing requirement of getting an average reward of 195.0 over 100 consecutive trials by the 600th generation.

A central limitation of this study was that it covered the implementation of the algorithm on only one test case and therefore the results can't be seen as indicative on the performance the algorithm could be expected to achieve in other test environments.

Recommendations for future study would be exploring more advanced reinforcement learning algorithms, such as Deep Q-Learning, which implements a neural network that takes a state and calculates Q-values for each action in that state instead of using a simple Q-table like in traditional Q-Learning.

References

Barber, D., (2012). *Bayesian Reasoning and Machine Learning*. Cambridge, UK: Cambridge University Press

Bennett, J., (2016). The Algorithm Behind the Curtain: Understanding How Machines Learn with Q-Learning (3 of 5). Retrieved January 1, 2019 from https://randomant.net/the-algorithm-behind-the-curtain-understanding-how-machines-learn-with-q-learning/

Bertsekas, D., P., (2011). *Dynamic Programming and Optimal Control 3rd Edition, Volume II*. Cambridge, MA, USA: Athena Scientific

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., (2016). OpenAI Gym. arXiv:1811.12560.

Francois-Livet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., (2018). An Introduction to Deep Reinforcement Learning. *Foundations and Trends in Machine Learning: Vol. 11, No. 3-4*.

Grinstead, C. M., Snell, J. S, (2006). *Introduction to Probability*. Providence, Rhode Island, US: American Mathematical Society.

ISO/IEC JTC 1, (1997). ISO/IEC 2382-31:1997(en) Information technology — Vocabulary — Part 31: Artificial intelligence — Machine learning. Retrieved January 1, 2019 from https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:-31:ed-1:v1:en

Jin, C., Allen-Zhu, Z., Bubeck, S., Jordan, M. I., (2018). Is Q-learning Provably Efficient? arXiv:1807.03765.

Konstantopoulos, T., (2009). Introductory lecture notes on Markov Chains and Random Walks. Retrieved January 1, 2019 from http://www.bioinfo.org.cn/~wangchao/maa/mcrw.pdf

Matiisen, T., (2015). Demystifying Deep Reinforcement Learning. Retrieved January 1, 2019 from https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

Melo, F. S., (n.d.). Convergence of Q-learning: a simple proof. Retrieved January 1, 2019 from http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf

OpenAI Gym Documentation, (n.d.). Retrieved January 1, 2019 from https://gym.openai.com/docs/

CartPole-v0 Documentation, (n.d.). Retrieved January 1, 2019 from https://gym.openai.com/envs/CartPole-v0/

Python 3.7.2 Documentation, (n.d.). Retrieved January 1, 2019 from https://docs.python.org/3/

Poole, D. L., Mackworth, A. K, (2017). *Artificial Intelligence: Foundations of Computational Agents, 2nd Edition*. Cambridge, UK: Cambridge University Press

Russell, S. J., Norvig, P., (2009). *Artificial Intelligence: A Modern Approach (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice Hall

Sigaud, O., Buffet, O., (2010) *Markov Decision Processes in Artificial Intelligence: MDPs, beyond MDPs and applications*. St Georges Road, London, UK: ISTE Ltd and John Wiley & Sons, Inc.

Silver, D., (2015). Advanced Topics 2015 (COMPM050/COMPGI13) Reinforcement Learning. Retrieved January 1, 2019 from http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html

Sutton, R. S., Barto, A. G., (2017). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: The MIT Press

Sutton, R. S., Barto, A. G., Anderson, C. W., (1983). *Neuronlike adaptive elements that can solve difficult learning control problems*. Piscataway, NJ, USA: IEEE Transactions on Systems, Man, and Cybernetics

Vilches, V. M., (2019). Basic Reinforcement Learning (RL). Retrieved January 1, 2019 from https://github.com/vmayoral/basic_reinforcement_learning

Watkins, J. C. H, Dayan, P., (1992). Technical Note: Q-Learning. Retrieved January 1, 2019 from http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf